Calcul distribué d'un extremum et du routage associé dans un réseau quelconque

Jean-Michel Hélary, Aomar Maddi, Michel Raynal

INRIA

CENTRE DE RENNES
IRISA

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France

Tél. (1) 39 63 55 11

Rapports de Recherche

N° 516

CALCUL DISTRIBUÉ D'UN EXTREMUM ET DU ROUTAGE ASSOCIÉ DANS UN RÉSEAU QUELCONQUE

Jean-Michel HELARY
Aomar MADDI
Michel RAYNAL

Avril 1986

IRISA

INSTITUT DE RECHERCHE EN INFORMATIQUE
ET SYSTÈMES ALÉATOIRES

Campus Universitaire de Beaulieu
35042-RENNES CÉDEX
FRANCE
Téléphone : 99 36 20 00
Télex : UNIRISA 950 473 F
Télécopie : 99 38 38 32

CALCUL DISTRIBUE D'UN EXTREMUM ET DU ROUTAGE ASSOCIE DANS UN RESEAU QUELCONQUE

JEAN-MICHEL HELARY, AOMAR MADDI, MICHEL RAYNAL

I.R.I.S.A

Publication Interne n° 289 - Mars 1986 - 36 pages

Abstract

Two distributed algorithms for election in a distributed system are presented. Both are original with respect to hypothesis and techniques involved. No particular assumption on the network topology is made, but connectivity; moreover, no process needs to know or learn the network global structure or size. In that sense, both algorithms are fully distributed and fundamentally different from those assuming a particular topology, known by all processes (such as complete network or ring).

Based upon the principle of "selective message extinction", the proposed algorithms extend to distributed contexts the sequential search strategies : "depth first" and "breadth first" respectively; to this end, specifically designed tools and techniques are introduced.

Résumé.

Deux algorithmes distribués réalisant une élection dans un système distribué sont présentés. Leur originalité réside dans les hypothèses qu'ils nécessitent et les techniques qu'ils utilisent. Aucune hypothèse particulière n'est faite sur le graphe qui modélise le réseau si ce n'est sa connexité; de plus aucun des processus n'a besoin de connaitre ni la structure globale du réseau ni même sa taille. Les algorithmes sont donc en ce sens entièrement distribués, et se distinguent fondamentalement des algorithmes qui s'appuient sur une topologie particulière connue de tous (tel qu'un anneau ou un maillage complet par exemple).

Fondés sur un même principe : l'extinction selective de messages, les algorithmes proposés généralisent au contexte distribué les stratégies de recherche séquentielle en profondeur et en largeur d'abord; des techniques et des outils spécialement adaptés sont introduits à cette fin.

CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE
(L. A. 227)
UNIVERSITÉ DE RENNES 1 I. N. S. A. DE RENNES

INSTITUT NATIONAL DE RECHERCHE
EN INFORMATIQUE ET EN AUTOMATIQUE
(LABORATOIRE DE RENNES)

SOMMAIRE

1 INTRODUCTION

La solution de certains problèmes de contrôle relatifs aux applications et aux systèmes distribués peut passer par la définition d'un rôle particulier que va jouer l'un (et un seul) des sites qui supportent l'application. C'est par exemple le cas dans les bases de données réparties où le problème du contrôle de la concurrence, dû à l'execution parallèle de transactions accédant des données distribuées, peut être résolu en associant à chaque ensemble de données (dont chacune est dupliquée sur les divers sites du réseau) un site particulier; le rôle de ce site est de contrôler les accès qu'effectuent les transactions de façon à assurer la cohérence mutuelle de ces données (respect des contraintes d'intégrité) et la convergence de leurs copies vers une même valeur [BER 81, GRA 78]; un tel site est généralement appelé site primaire. La désignation d'un processus destiné à coordonner les diverses activités dans un système distribué constitue un autre exemple du même problème [LEL 78].

Il est alors essentiel de savoir définir un tel site et de donner à chacun des autres sites les moyens de communiquer avec lui. Ce site est généralement choisi en fonction d'un critère de poids et il est d'usage de choisir celui dont le poids est un extremum en considérant comme poids d'un site son identité représentée par un numéro. Une fois ce choix effectué il est nécessaire d'une part de communiquer l'identité de ce site à tous les autres sites et d'autre part d'établir des chemins de communication de ces sites vers le site choisi.

Une première solution consisterait à effectuer le choix d'un site, à établir les chemins correspondants de manière statique et à incorporer ces éléments dans les définitions des sites lors de la génération du système ou de l'application. L'absence de souplesse de cette solution est rédhibitoire et a conduit vers la recherche de solutions dynamiques.

Ces solutions réalisent ce qu'il est convenu d'appeler un algorithme distribué d'élection : tous les sites (nous confondrons dans ce qui suit les termes site et processus, un site se présentant en effet comme une seule activité du point de vue de l'élection) sont initialement candidats à jouer le rôle particulier; au terme de l'algorithme seul celui dont l'identité est

la plus grande (ou la plus petite) peut effectivement le jouer, les autres pouvant alors solliciter ce dernier.

Les algorithmes proposés jusqu'alors ne s'intéressent, à notre connaissance, qu' au cas où le maillage du réseau d'une part est connu de tous (ce qui élimine le problème de l'établissement des chemins) et d'autre part est restreint à être soit un anneau (uni- ou bi-directionnel) [CHA 79, FRA 82, DOL 82, HIR 80, PET 82] soit un maillage complet [GAR 81, KOR 84]. Les meilleurs de ces algorithmes ont , dans les deux cas, une complexité $O(n\log n)$ où n est le nombre de sites du réseau (c'est la complexité minimale possible dans le cas d'un anneau [PAC 84]).

Nous présentons dans cet article des algorithmes distribués d'élection qui d'une part calculent un extremum dans un réseau quelconque, et d'autre part définissent une arborescence couvrante de ce réseau (dont la racine est le processus qui a pour identité l'extremum) permettant à tous les processus de communiquer via des chemins uniques avec le processus associé à la racine.

L'article est divisé en six parties. Dans la seconde on précise les hypothèses et l'intérêt qu'elles présentent; dans la troisième les principes qui sous-tendent les algorithmes sont exposés. La quatrième partie présente un premier algorithme assimilable à une exploration distribuée en profondeur d'abord alors que l'algorithme présenté dans la cinquième partie s'appuie sur une exploration parallèle (en largeur). Enfin la conclusion synthétise les résultats obtenus et dégage l'originalité de la démarche proposée.

2 HYPOTHESES

2.1 Hypothèses sur le réseau

Les processus ne peuvent s'échanger de l'information qu'à l'aide de messages véhiculés sur un réseau; il n'y a donc aucune mémoire partagée entre les processus.

Le réseau de communication, modélisé par un graphe, est connexe mais quelconque. Les liaisons entre deux processus sont bi-directionnelles, en d'autres termes le graphe est non orienté. Elles sont dotées des propriétés

de comportement suivantes : les messages ne se perdent pas et sont délivrés au bout d'un temps fini après leur émission, de plus ils ne sont pas altérés. Toutefois leur déséquencement est possible : entre deux processus P_i et P_j reliés par une ligne de communication l'ordre dans lequel P_i reçoit les messages que lui a envoyé P_j n'est pas nécessairement le même que l'ordre d'émission; en d'autres termes il peut y avoir dépassement de messages sur les lignes.

2.2 Hypothèses sur les processus

Le réseau comporte n processus qui ont des identités distinctes. Un processus ne connait initialement que ses voisins directs, de plus il les connait par leurs identités. Au cours du calcul distribué un processus quelconque n'apprendra jamais la structure globale du réseau, ni même le nombre total n de processus participants : sa connaissance topologique se limitera toujours à ses voisins directs.

2.3 Intérêt des hypothèses

Ces hypothèses sont particulièrement intéressantes car elles facilitent la résistance aux pannes et les problèmes posés par la génération de systèmes distribués. La définition d'un site n'inclut alors pas d'informations globales; en conséquence la configuration du système peut être modifiée sans que cela entraine une nouvelle définition de tous les processus et sans que le système soit tributaire d'une topologie particulière; seuls les processus dont le voisinage a été modifié doivent être redéfinis et donc générés à nouveau.

Dans le cas où un processus ne connait son voisinage que par l'identité des liaisons (canaux) qui le connectent à ses voisins, l'identité de ceux-ci peut être acquise grâce à un protocole simple : deux messages par canal (un dans chaque sens) permettent l'échange des identités entre tout couple de processus voisins [SEG 83].

3 PRINCIPE DES SOLUTIONS

Dans un contexte séquentiel une réponse au problème consisterait à rechercher dans le graphe associé au réseau le sommet qui possède la plus grande identité puis à construire, avec ce processus pour racine, une arborescence recouvrante qui établirait les chemins entre un processus quelconque et la racine.

Dans un contexte distribué la solution dans laquelle tous les processus enverraient à l'un d'entre eux leur identité et celles de leurs voisins n'est pas possible car (outre le fait qu'elle ne respecte pas les hypothèses sur la localité des informations) elle nécessite la définition préalable d'un processus unique qui collecterait toutes les informations. Ce qui reviendrait à résoudre préalablement un problème ...d'élection.

Une des caractéristiques essentielles du calcul distribué, qui fait une de ses originalités et de ses difficultés, réside dans le fait qu'un algorithme qui produit un unique résultat, peut être démarré "simultanément" par un, plusieurs, voire tous les processus. Le résultat doit à priori être indépendant du nombre de processus (au moins un) qui ont activé l'algorithme.

Le principe sur lequel reposent les deux algorithmes proposés est le suivant :

pour un processus P_i, lancer une élection consiste à lancer une exploration du réseau dont le but est de visiter tous les processus en établissant des chemins de ceux-ci vers lui-même. Comme l'algorithme peut être démarré par un ou plusieurs processus, une ou plusieurs explorations peuvent être simultanément en cours. Au terme de l'algorithme il est nécessaire que l'exploration issue du processus doté de la plus grande identité d'une part se soit terminé correctement (i.e ait visité tous les processus en établissant l'arborescence recouvrante) et d'autre part que celle-ci soit la dernière exploration prise en compte par les autres processus (i.e que ceux-ci mémorisent les liaisons auxquelles ils participent dans l'arborescence construite). On peut identifier chaque exploration par son poids, c'est à dire l'identité du processus dont elle est issue; les deux

objectifs ci-dessus seront atteints dès lors que : tout processus P_i visité par une exploration de poids j la stoppe sans la prendre en compte s'il a déjà été visité par une exploration de poids k avec $k > j$, ou bien, n'ayant encore jamais été visité et n'ayant pas lancé d'exploration il trouve $i > j$, auquel cas il lance une exploration; dans les cas contraires il prend en compte l'exploration de poids j et la fait progresser.

Deux types d'explorations issues d'un processus P_i sont envisageables. Lorsqu'un processus est visité par une telle exploration et la prend en compte il peut la faire progresser soit vers l'un de ses voisins (le plus grand par exemple) soit vers tous : c'est ce qui distingue les deux algorithmes proposés. Comme on le voit ils étendent au contexte distribué les techniques séquentielles d'exploration en profondeur ou en largeur d'abord.

Nous allons présenter dans le détail le premier algorithme, en donner la preuve et une analyse de complexité. Pour le second nous nous limiterons à l'algorithme lui-même, la preuve étant analogue.

4 ALGORITHME 1: EXPLORATION EN PROFONDEUR

4.1 L'algorithme

4.1.1 Les étapes d'une exploration

Une exploration E_k de poids k (issue de P_k) peut être vue comme activité séquentielle : une séquence de messages (de poids k) est engendrée; chaque message est traité par le processus qui le reçoit et celui-ci, à son tour, peut éventuellement émettre un nouveau message de poids k. A chaque instant de l'exploration, un seul message de poids k est en transit sur une ligne ou en traitement par un processus. On peut donc parler des étapes de l'exploration, matérialisées -par exemple- par la réception d'un message μ par un processus P_i (message et processus courants). Selon le contexte du processus P_i et l'information transportée par le message μ, on distingue quatre types d'étapes.

4.1.1.1 Etape d'extinction

- ou bien P_i a déjà été atteint par une exploration de poids supérieur à k (ou a déjà lancé sa propre exploration E_i si $i > k$),

- ou bien P_i n'a pas encore été atteint, mais $i > k$.

Dans un cas comme dans l'autre, l'exploration E_k est éteinte par P_i; de plus, dans le deuxième cas, P_i lance l'exploration de poids i.

4.1.1.2 Etape de conclusion

Dans cette étape et les deux suivantes, on suppose que E_k est l'exploration de poids le plus fort ayant jamais atteint P_i; *max* désignera la plus grande identité de l'ensemble des processus du réseau.

L'information de contrôle amenée par le message indique à P_i que tous les autres processus ont été atteints par l'exploration. Dans ce cas, E_k s'arrête et on a alors nécesssairement : $k = max$; P_i peut donc conclure. L'exploration E_{max} a établi une arborescence recouvrant le réseau (modélisé par un graphe G non orienté connexe) et les autres processus sont informés de la fin de l'algorithme par un signal diffusé sur les branches de cette arborescence.

4.1.1.3 Etape de génération

P_i possède des voisins non encore atteints par E_k. Il met alors à jour les liaisons relatives à l'exploration E_k, et engendre un nouveau message d'exploration vers un de ses voisins non atteints (par exemple le plus grand).

4.1.1.4 Etape de renvoi

Tous les voisins de P_i ont été atteints par E_k, mais l'information de contrôle amenée par le message indique que des processus du réseau n'ont pas encore été atteints par E_k. P_i engendre alors un message permettant de remonter vers un processus non encore atteint.

Dans ce qui suit nous dirons qu'un processus P_i est "visité" par l'exploration E_k losqu'il est atteint par cette exploration et ne l'éteint

pas. De plus la fonction *maximum* appliquée à un ensemble z d'identités de de processus délivre la plus grande d'entre elles.

4.1.2 Les messages utilisés

La mise en oeuvre du principe énoncé précédemmment nécessite trois types de messages distincts :

(i) les messages du type *explorer* sont de la forme suivante :

explorer(k,z,s)

Emis lors d'une étape de génération, ils réalisent l'exploration des processus du réseau pour le compte du processus d'identité k. Le champ z contient les identités des processus visités par ce message (on doit avoir $k = maximum(z)$), le champ s définit l'ensemble des processus voisins immmédiats des processus de z qui n'ont pas encore été atteints par cette exploration (la condition $s \neq \emptyset$ indique donc que l'exploration lancée par P_k n'est pas terminée) . s est donc un sous-ensemble des processus non atteints par E_k.

(ii) les messages du type *rebrousser* , émis lors d'une étape de renvoi, permettent à une exploration qui n'a pas atteint tous les processus (on a alors nécessairement $s \neq \emptyset$) mais qui ne peut progresser à partir du processus actuellement visité, de repartir à partir d'un processus visité qui possède un voisin appartenant à s. Les champs d'un tel message sont les mêmes que ceux du message de type *explorer*.

(iii) les messages du type *conclure* servent à signaler aux processus que l'exploration lancée par P_{max} a atteint tous les processus; ils réalisent en conséquence la terminaison de l'algorithme.

4.1.3 Contexte d'un processus

Tous les processus ont les mêmes définitions de contextes et executent les mêmes textes d'algorithme. L'algorithme est symétrique en ce sens-là [RAY 85]. Dans tout ce qui suit on se place du point de vue d'un processus quelconque, que nous appelerons P_i.

const $voisins_i$ **initialisée à** {identités des voisins de P_i};
Cet ensemble définit le voisinage de P_i, qui est sa seule connaissance sur le réseau.

var $état_i$: *(initial, candidat, battu, élu)* **initialisé à** *initial*;
Cette variable donne l'état courant de P_i; à la terminaison de l'algorithme on doit avoir :
$\exists! j : (état_j = élu \wedge \forall\ k \neq j :\ état_k = battu)$.

$pgvu_i$: **entier initialisé à** *i*;
Donne l'identité de la plus grande exploration qu'ait jamais vu P_i; à la fin on aura $pgvu_i$ = *max* (identité maximum).

$pred_i$: **entier initialisé à** *nil*;

$succ_i$: **ensemble d'entier initialisé à** $\emptyset$;
Ces deux variables serviront à placer P_i dans l'arborescence construite (*nil* désigne une valeur qui n'est l'identité d'aucun processus).
(On a nécessairement $succ_i \cup \{pred_i\} \subseteq voisins_i$).

4.1.4 Comportement de P_i

Le comportement auquel obéit un processus est donné (à la manière de [RAY 85]) sous la forme d'un automate associant un traitement à chaque événement qui peut survenir (réception d'un message d'un processus voisin ou décision locale de lancer une élection). La définition suivante permet d'en condenser l'écriture.

```
procédure lancer_exploration;
début
        soit x = maximum(voisins);
        état_i := candidat;
        pred_i := nil;
        succ_i := {x};
        envoyer explorer(i,{i},voisins-{i}) à P_x
 fin
```

```
lors de décision de lancer une élection
faire
    si état_i = initial alors lancer_exploration fsi
fait


lors de réception de explorer(k,z,s) depuis P_j
faire
   cas pgvu_i > k → si état_i =initial alors lancer_exploration fsi
       pgvu_i < k → état_i := battu;
                    pgvu_i := k;                                        (a1)
                    pred_i := j;
                    soit y = voisins_i - z;
                    cas y = ∅ → succ_i := ∅;
                         cas s = ∅ → envoyer conclure à P_j             (a2)
                              s ≠ ∅ → envoyer rebrousser(k,zU{i},s) à P_j
                         fcas
                        y ≠ ∅ → soit x = maximum(y);
                                succ_i := {x};
                                envoyer explorer(k,zU{i}, sUy-{x}) à P_x
                  fcas
   fcas
fait


lors de réception de rebrousser(k,z,s) depuis P_j
faire
       si pgvu_i = k alors
            soit y = voisins_i ∩ s;                                     (a3)
            cas y = ∅ → envoyer rebrousser(k,z,s) à P_pred_i
                y ≠ ∅ → soit x = maximum(y);
                        succ_i := succ_iU{x};
                        envoyer explorer(k,z,s-{x}) à P_x               (a4)
            fcas
      fsi
fait


lors de réception de conclure depuis P_j
faire
     si pgvu_i = i alors état_i := elu fsi;
     ∀ x∈ (succ_i Upred_i)- {j} :  envoyer conclure à P_x               (a5)
fait
```

4.2 Preuve

4.2.1 Propriétés

La compréhension et la preuve de l'algorithme sont facilitées si l'on met en évidence un certain nombre de propriétés ponctuelles ou invariantes, relatives à une exploration donnée; celles-ci concernent :

- la nature et le contenu des champs des messages,
- l'arborescence construite par une exploration sur le graphe G associé au réseau.

(Pour ne pas alourdir la présentation, la démonstration de ces propriétés est donnée à l'annexe 1).

4.2.1.1 Les messages de type *explorer* reçus par un processus

(M1) : Le premier message de poids k éventuellement reçu par un processus P_i avec $i \neq k$, est nécessairement du type *explorer*.

(M2) : P_k ne peut recevoir aucun message *explorer* de poids k.

(M3) : $\forall\ i \neq k$, P_i reçoit au plus un message *explorer* de poids k.

4.2.1.2 Arborescence d'exploration

Une exploration E_k, de poids k, établit un sous-graphe partiel de G, désigné par $A_k = (X_k, \Gamma_k)$, avec :

$X_k = \{ P_i \mid P_i$ a été visité par $E_k \}$

$\Gamma_k = \{(P_i, P_j) \in X_k^2 \mid P_i$ a envoyé à P_j un message *explorer* de poids $k\}$

Autrement dit, lors d'une étape de E_k, le processus courant et la ligne par laquelle le message courant lui est parvenu (orientée par le sens de circulation de ce message) sont ajoutés à A_k si et seulement si, d'une part, le message courant est du type *explorer* et, d'autre part, le processus courant n'éteint pas l'exploration.

Au graphe A_k est associé l'invariant suivant :

(A1) : A chaque étape de l'exploration de poids k , A_k définit une arborescence de racine P_k dans laquelle, pout tout P_i vérifiant $pgvu_i = k$:

- $pred_i$ désigne le père de P_i,
- $succ_i$ désigne l'ensemble des fils de P_i.

Remarque : Pour les processus $P_i \in X_k$ qui ont ensuite été atteints par une exploration de poids plus élevé (on a alors $pgvu_i > k$) les liens dans l'arborescence ne sont plus mémorisés; seules les liens relatifs à l'arborescence de poids $pgvu_i$ sont mémorisés par P_i.

Pour tout $P_j \in X_k$, nous désignerons par γ_{kj} le chemin unique de A_k, reliant la racine P_k à P_j.

4.2.1.3 **Invariants relatifs à l'information de contrôle**

Pour tout message de poids k (*explorer(k,z,s)* ou *rebrousser(k,z,s)*) émis de P_i vers P_j, les champs z et s vérifient :

(P1) : z est l'ensemble des identités des processus ayant reçu un message *explorer* de poids k et n'ayant pas éteint l'exploration lorsque celle-ci les atteignit.

(P2) : $k = maximum(z)$.

(P3) : s est égal à l'ensemble des identités des processus non atteints par E_k, voisins des processus situés sur γ_{kj}.

(P4) : Si $l \in z$ et si P_l possède des voisins non atteints, alors P_l est situé sur γ_{kj}.

Remarque : Il en résulte de (P3) et (P4) que s a bien la propriété annoncée au 4.1.2 i) : c'est l'ensemble des processus non atteints, voisins des processus déjà visités par E_k.

L'annexe 1 démontre les propriétés : M1 à M3, A1 et P1 à P4.

4.2.2 Arrêt d'une exploration

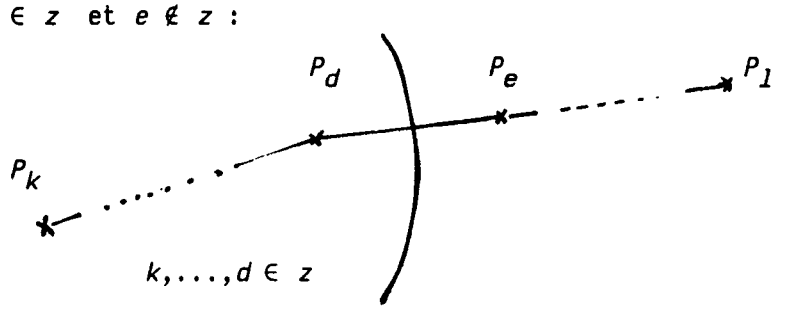
4.2.2.1 Proposition 1

Une exploration atteint l'étape de conclusion si et seulement si tous les processus du réseau sont visités par cette exploration.

Démonstration

supposons que E_k atteigne l'étape de conclusion lors de la réception du message μ par P_i; les contextes de μ et P_i vérifient alors :

$voisins_i \subseteq z$ et $s = \emptyset$, et μ est du type *explorer* (ligne a2 de l'algorithme). Supposons qu'il existe un processus P_l tel que $l \notin z$. D'après la connexité du réseau, il existe dans G une chaine de P_k à P_l; comme $k \in z$, cette chaine passe par une arête sortant de z, soit (d,e) cette arête avec $d \in z$ et $e \notin z$:



P_d possède donc un voisin non atteint P_e. D'après (P3) : P_d n'est pas sur le chemin γ_{ki} puisque $s = \emptyset$ et $voisins_i \subseteq z$ (les voisins de P_i sont atteints). La propriété (P4) est donc contredite par P_d qui ne peut à la fois être et ne pas être sur γ_{ki}. QED.

4.2.2.2 Proposition 2

Une exploration de poids k, si elle est lancée, est nécessairement stoppée au bout d'un nombre fini d'étapes :

- par une étape d'extinction si $k < max$,
- par l'étape de conclusion si $k = max$. A_{max} est alors une arborescence recouvrant le graphe G.

Démonstration

Lors d'une exploration E_k, l'étape courante (réception de μ par P_i) peut être du type :

(i) extinction : si $k < max$, la proposition est démontrée. Si $k = max$, une telle étape ne sera pas rencontrée car, $\forall$ P_i on a $pgvu_i \leq max$ ($pgvu_i$ est le maximum des poids des explorations ayant visité P_i).

(ii) conclusion : Si $k < max$, une telle étape ne sera pas rencontrée, car d'après la proposition 1, cela signifierait que tous les processus sont visités par l'exploration. Or, il existe au moins un processus P_j avec $j > k$ qui, lorsqu'il est atteint, vérifie $pgvu_j \geq j > k$ et donc, éteint E_k. Si $k = max$, la proposition est démontrée.

(iii) génération : P_i envoie un message *explorer* à l'un de ses voisins, non encore atteint par E_k (propriété M2 et M3). Ce voisin :
- soit éteint E_k (ce qui implique $k < max$) et alors la proposition est démontrée,
- soit entre dans X_k, et donc le nombre de sommets visités par E_k augmente de un.

(iv) renvoi : tous les voisins de P_i ont été visités, mais $s \neq \emptyset$. E_k engendre alors une séquence de messages *rebrousser*, provoquant une remontée sur la branche γ_{ki}. Cette branche étant de longueur finie, la remontée est stoppée au bout d'un nombre fini d'étapes, soit :
- par la rencontre d'un processus P_j pour lequel $pgvu_j > k$, ce qui implique $k < max$ (depuis qu'il a été visité par l'exploration E_k, P_j a été visité par une exploration de poids supérieur à k), la propriété est alors démontrée.
- par la rencontre de P_l tel que $voisins_l \cap s \neq \emptyset$ (ligne a3) : d'après (P3), ceci se produira nécessairement, au plus tard à la racine P_k de l'exploration E_k (sauf si E_k est éteinte). L'exploration est alors relancée par une étape de génération (ligne a4).

Ainsi, tant que l'exploration E_k n'est pas stoppée, le nombre de sommets qu'elle visite croît strictement, et un nombre fini d'étapes séparent deux étapes de génération. Comme par ailleurs : $k = maximum(z)$ (propriété P2) il en résulte que :

. si $k < max$, un sommet P_j tel que $pgvu_j \geq j > k$ sera atteint au bout d'un nombre fini d'étapes (extinction),

. si $k = max$, tous les sommets seront visités au bout d'un nombre fini d'étapes (conclusion). QED.

4.2.3 Terminaison et correction partielle

4.2.3.1 Theorème 1 : Terminaison.

L'algorithme se termine au bout d'un temps fini.

Démonstration

(i) D'après les hypothèses sur le réseau, l'intervalle de temps séparant deux étapes d'une exploration donnée est fini (acheminement d'un message sur une ligne en un temps fini).

(ii) D'autre part, l'exploration E_{max} est lancée au bout d'un nombre fini d'étapes. En effet,par hypothèse, au moins un processus P_k lance une exploration de poids k. Si $k = max$, ce résultat est démontré. Sinon, d'après la proposition 2, l'exploration E_k sera éteinte au bout d'un nombre fini d'étapes, par un processus P_j tel que $pgvu_j = k_1 > k$. Si $k_1 \neq j$ l'exploration E_{k_1} a déjà été lancée, si $k_1 = j$ elle est lancée à l'extinction de E_k. L'exploration de poids k_1 est donc lancée au plus tard à l'extinction de E_k. Réitérant ce raisonnement, on construit une suite strictement croissante $k < k_1 < k_2 < \ldots$ qui converge donc vers *max* en un nombre fini d'étapes.

(iii) D'après le point précédent (ii), l'exploration E_{max} est lancée après un nombre fini d'étapes et d'après la proposition 2 elle s'arrête en un nombre fini d'étapes.

(iv) Enfin , l'étape de conclusion est atteinte par E_{max} (propositions 1 et 2) : les messages *conclure* parcourent alors l'arborescence A_{max} (construite par E_{max}), chacune de ses arêtes étant visitée une et une seule fois (ligne a5).

Les 4 points démontrent donc la terminaison de l'algorithme. QED.

4.2.3.2 **Théorème 2 : Correction partielle.**

A la fin de l'algorithme, tous les processus sont informés de la terminaison et connaissent :

- l'identité du processus élu, égale à *max* la plus grande identité; elle est mémorisée dans $pgvu_i$, $\forall P_i$.
- le routage permettant d'y accéder, mémorisé dans $pred_i$, $\forall P_i$.

Démonstration

Tous les processus ayant été visités par l'exploration de poids *max*, on a $\forall P_i$: $pgvu_i = max$. (ligne α1 et propositions 1 et 2).

D'autre part, cette exploration a établi l'arborescence A_{max} de racine P_{max} (proposition 2) telle que $pred_i$ est le père de P_i dans cette arborescence (invariant A1); de plus la réception par P_i d'un message *conclure* lui signale la terminaison de l'algorithme (ligne α5). QED.

4.3 **Analyse de complexité**

Une exploration de poids k (issue de P_k) est stoppée (proposition 2) :

- si $k < max$, au plus tard lorsqu'un processus P_l, tel que $l > k$, est atteint par un message *explorer* de poids k,
- si $k = max$, lorsque tous les processus sont visités.

De plus, P_k envoie le premier message *explorer* vers son voisin de plus fort poids (cf. la procédure lancer_exploration). Il en résulte que :

- Si $k < max$ et P_k n'est pas un maximum local, alors E_k engendre un message *explorer* et aucun message *rebrousser*.
- Si $k < max$ et P_k est un maximum local alors E_k engendre au plus k' messages *explorer*, et donc au plus k' messages *rebrousser* (puisqu'un message *rebrousser* de poids k ne peut parcourir une ligne que si un message *explorer* a parcouru la même ligne en sens inverse), k' étant la position de P_k dans l'ensemble ordonné des identités ($k' \in 1..n$).
- Si $k = max$, alors E_{max} engendre exactement $n-1$ messages *explorer* et au plus $n-1$ messages *rebrousser*.

Chaque exploration est lancée au plus une fois (exactement une fois pour

P_{max}).

Enfin le nombre de messages *conclure* est égal exactement à *n-1*.

On obtient donc les résultats suivants.

4.3.1 cas le plus favorable

Ce cas est obtenu lorsque P_{max} est seul à lancer une exploration; on a alors :

- *n - 1* messages *explorer*
- *0 à n - 1* messages *rebrousser* (selon la topologie du réseau; la valeur minimale 0 est obtenue lorsqu'il existe une chaine dans le réseau incluant tous les processus dans l'ordre décroissant de leurs identités. C'est le cas par exemple d'un anneau, d'un réseau complet et d'un réseau linéaire lorsque P_{max} est en bout de ligne; la valeur maximale *n-1* est obtenue dans le cas d'un réseau en étoile autour de P_{max}).
- *n*-1 messages *conclure*.

4.3.2 Cas le plus défavorable

Dans ce cas on a :

$$\text{au plus } \left(\sum_{k'=1}^{n-1} k'\right) + (n-1) \text{ messages du type } explorer,$$

autant de messages *rebrousser* et *n-1* messages *conclure* soit :

$$2\left(\left(\sum_{k'=1}^{n-1} k'\right) + (n-1)\right) + n-1 = (n-1)(n+3) \text{ messages, tous types confondus.}$$

La complexité est donc $O(n^2)$ dans le pire des cas.

L'annexe 2 étudie le nombre de messages de cet algorithme (qui rappellons-le est indépendant de la topologie) dans les deux cas particuliers que sont le réseau complet et l'anneau.

5 ALGORITHME 2 : EXPLORATION PARALLELE

5.1 Arbre de contrôle

Dans cet algorithme l'exploration issue d'un processus P_k est lancée en parallèle vers tous ses voisins. Comme précédemment un processus ne laisse progresser vers ses voisins la composante de l'exploration qui l'atteint que si le poids k de celle-ci est supérieur au poids de la dernière exploration qu'il a pris en compte. Cette condition assure que seule l'exploration lancée par P_{max} pourra être prise en compte par tous les processus. L'arborescence recouvrante de racine P_{max} est construite par les messages relatifs à l'exploration correspondante.

Pour obtenir une arborescence recouvrante, il est nécessaire que tout processus autre que P_{max} ait un seul prédécesseur et que deux processus distincts aient des ensembles de successeurs disjoints. Ceci n'est pas implicitement réalisé par la circulation des messages relatifs à une exploration donnée. Si comme précédemment la prise en compte par P_i d'un message d'exploration peut permettre de définir son prédécesseur, il est nécessaire, pour établir ses successeurs, d'introduire un contrôle adéquat. Plusieurs protocoles permettent de résoudre ce problème [SEG 83]. Tout processus P_i, lorsqu'il reçoit un message relatif à une exploration qu'il prend en compte pour la première fois, mémorise l'émetteur P_j du message comme étant son prédécesseur, propage l'exploration vers ses voisins, et attend d'avoir reçu des acquittements de tous ceux-ci, avant d'emettre l'acquittement vers P_j; dans le cas où il reçoit un message relatif à l'exploration déjà prise en compte, il l'acquitte immédiatement en indiquant à P_j de ne pas l'inclure dans ses successeurs.

Cette technique est abondamment utilisée dans de nombreux algorithmes où il s'avère nécessaire de construire une arborescence pour contrôler un calcul; citons à titre d'exemple : la terminaison distribuée [DIS 80, FRR 82, MIS 82] la détection de l'interblocage [CHA 83, HMR 86], et le calcul distribué sur les graphes [CHA 82, MIC 82].

5.2 L'algorithme

5.2.1 Les messages utilisés

Les messages du type *rebrousser* sont maintenant inutiles à cause du parallélisme de l'exploration; il en est de même du champ *s* du message *explorer*. Les messages de ce type présentent la forme suivante :

explorer(k,z)

où *k* est le poids du processus qui a lancé l'exploration correspondante et où *z* désigne l'ensemble des processus dont on est sûr qu'ils ont été ou seront visités par des messages relatifs à cette exploration. Le fait de placer (selon la technique proposée dans [HMR 86]) dans *z* non seulement les processus visités mais aussi des informations sur le voisinage du processus émetteur va permettre de diminuer le nombre de messages échangés.

Les acquittements relatifs aux messages d'exploration sont véhiculés par les messages du type *acquitter* qui présentent deux champs :

acquitter(k,x)

Dans un tel message émis par P_i : *k* désigne l'exploration considérée et *x* permet au récepteur de savoir s'il doit considérer P_i comme un de ses successeurs dans l'arborescence (*x = terminé*) ou non (*x = dejàvu*).

La réception par P_{max} de tous les acquittements en provenance de ses voisins indique la terminaison de l'algorithme.

5.2.2 Contexte d'un processus

La constante $voisins_i$ et les variables $état_i$, $pgvu_i$, $pred_i$ et $succ_i$ ont la même signification qu'au paragraphe 4.1.3.

la variable **var** $nbacq_i$ **(entier non négatif initialisé à 0)** permet à P_i de mémoriser le nombre d'acquittements attendus et de contrôler l'envoi de l'acquittement vers son prédécesseur dans l'arborescence construite.

5.2.3 Comportement de P_i

procédure lancer_exploration;
debut
$état_i$:= *candidat*;
$pred_i$:= *nil*;
$succ_i$:= $voisins_i$;
$nbacq_i$:= cardinal($succ_i$);
$\forall x \in succ_i$: **envoyer** *explorer*(i,$voisins_i$) **à** P_x
fin

```
lors de décision de lancer une élection
faire
      si état_i = initial alors lancer_exploration fsi
fait


lors de réception de explorer(k,z) depuis P_j
faire
   cas
     pgvu_i > k→ si état_i = initial alors lancer_exploration fsi
     pgvu_i = k → envoyer acquitter(z,déjàvu) à P_j
     pgvu_i < k → état_i := battu;
                  pgvu_i := k;
                  pred_i := j;
                  succ_i := voisins_i - z;
                  cas
                    succ_i=∅ → envoyer acquitter(z,terminé) à P_j
                    succ_i≠∅ → nbacq_i := cardinal(succ_i);
                               ∀x∈succ_i : envoyer explorer(k,zUsucc_i) à P_x
                  fcas
   fcas
fait


lors de réception de acquitter(k,x) depuis P_j
faire
     si pgvu_i = k alors
          si x = déjàvu alors succ_i := succ_i - {j} fsi;
          nbacq_i :=nbacq_i -1;
          si nbacq_i = 0 alors
                cas k = i → état_i := élu
                    k ≠ i → envoyer acquitter(x,terminé) à P_pred_i
                fcas
     fsi  fsi
fait
```

6 CONCLUSION

Les deux algorithmes présentés s'appuient sur un principe général qu'il est important de caractériser : l'extinction sélective de messages. Dans notre cas une exploration lancée par un processus sera ou non stoppée en fonction de son poids et de celui que mémorise le processus atteint. Ce principe, bien que non explicitement formulé, est souvent rencontré dans de nombreux algorithmes distribués de contrôle et notamment dans la résolution de problèmes où l'arrivée d'un message doté d'un attribut tel que sa date d'émission (mis en oeuvre par une estampille ou un numéro de séquence) rend caduques les messages qui, émis avant lui (et donc dotés d'une estampille inférieure), arriveront après lui. Citons à titre d'exemples :[CHA 83] en ce qui concerne la détection de l'interblocage, [SCH 84] pour la diffusion fiable et [HPR 86] pour l'exclusion mutuelle dans un réseau quelconque.

Les deux algorithmes proposés constituent deux mises en oeuvre de ce principe général, qui généralisent au contexte distribué les stratégies de recherche séquentielle que sont la "profondeur d'abord" et la "largeur d'abord". D'autres algorithmes distribués réalisent de telles traversées de réseaux mais supposent que, seul un des processus, qui sera la racine de l'arborescence construite, a l'initiative du lancement [AWE 85,CHE 83]. Les algorithmes qui sont ici proposés (dont la complexité en nombre de messages est identique à ceux-ci) présentent un double avantage. Tout d'abord les hypothèses faites sont plus faibles : aucun processus n'a besoin de connaitre la structure du réseau, ni même sa taille. Ensuite l'arborescence construite n'est pas quelconque : c'est le processus de poids maximum qui est particularisé et c'est l'arborescence correspondante qui est construite.

Sur le plan de l'algorithmique distribuée, les deux algorithmes proposés utilisent une technique de contrôle originale mise en oeuvre par les champs *z* et *s* des messages de type *explorer*.

Ces champs véhiculent une information de contrôle permettant de réduire le nombre de messages échangés d'une part en évitant d'effectuer des parcours

totalement aveugles (utilisation de *z* et *s* dans le premier algorithme et de *z* dans le second) et d'autre part en permettant de détecter la fin des parcours "au plus tôt" (utilisation de *s* dans le premier algorithme). (Il est en outre intéressant de noter que, en ce qui concerne la technique d'exploration en profondeur, l'information de contrôle *s* joue dans le contexte distribué le même rôle que la pile dans le contexte séquentiel). Les informations de contrôle sont d'un emploi général (elles sont appliquées dans [HMR 86] à la détection de l'interblocage dans les systèmes distribués); elles constituent un premier pas vers des techniques de parcours "intelligentes" dans de tels systèmes.

Signalons enfin que la conception de ces algorithmes a été facilitée par l'emploi d'un outil adéquat : le simulateur pour la validation de protocoles VEDA developpé au CNET de LANNION [JAR 85].

REFERENCES

[AWE 85] AWERBUCH B.
A new distributed depth-first search Algorithm.
Inf. Proc. Letters, 20,(April,1985), pp.147-150.

[BER 81] BERNSTEIN P.A., GOODMAN N.
Concurrency control in distributed data base systems.
ACM, Computing Surveys, Vol.13,2, (june,1981), pp.185-201.

[CHA 82] CHANDY K.M., MISRA J.
Distributed Computing on graphs :Shortest paths Algorithms.
Comm. ACM, Vol.25,11, (November 1982), pp.833-837.

[CHA 83] CHANDY K.M., MISRA J., HAAS J.
Distributed deadlock detection.
ACM TOCS, Vol. 1,2,(May 1983), pp.144-156.

[CHA 79] CHANG E.J., ROBERTS R.
An improved algorithm for decentralized extrema-finding in circular configurations of processors.
Comm. ACM, Vol. 22,5, (May 1979), pp. 281-283.

[CHE 83] CHEUNG T.
Graph traversal techniques and the maximum flow problem in distributed computation.
IEEE Trans. on soft. Eng., Vol. SE9,4, (july 1983), pp. 504-512.

[DIS 80] DIJKSTRA E.W., SHOLTEN C.S.
Terminaison detection for diffusing computations.
Inf. Proc. Letters, Vol. 11,1 (August 1980), pp.1-4.

[DOL 82] DOLEV D., KLAWE M., RODEH M.
An O(nlogn) unidirectional distributed algorithm for extrema finding in a circle.
Journal of Algorithms, Vol. 3, (1982), pp.245-260.

[FRR 82] FRANCEZ N., RODEH M.
Achieving distributed termination without freezing.
IEEE Trans. on Soft. Eng., Vol. SE8,3, (May 1982), pp. 287-292.

[FRA 82] FRANKLIN W.R.
On an improved algorithm for decentralized extrema-finding in circular configurations of processors.
Comm. ACM, Vol. 25,5, (May 1982), pp. 336-337.

[GAR 81] GARCIA-MOLINA H.
Elections in a distributed computing system.
IEEE Trans. on Computers, Vol C31,1, (january 1981), pp. 48-59.

[GRAY 78] GRAY J.N.
Notes on data base operating systems.
LNCS n°68, Springer-Verlag, (1978), pp. 393-481.

[HMR 86] HELARY J.M., MADDI A., RAYNAL M.
Controlling knowledge transfers in distributed algorithms : Aplication to deadlock detection.
Rapport de recherche IRISA N°278 (janvier 1986), 28p.

[HPR 86] HELARY J.M., PLOUZEAU N., RAYNAL M.
A distributed algorithm for mutual exclusion in an arbitrary network.
Rapport de recherche IRISA N°281, (janvier 1986), 15p.

[HIR 80] HIRSCHBERG D.S., SINCLAIR J.B.
Decentralized extrema finding in circular configurations of processors.
Comm. ACM, Vol. 23,11, (november 1980), pp.627-628.

[JAR 85] JARD C., MONIN J.F., GROZ R.
VEDA : a software simulator for the validation of protocol specifications.
COMNET 1985, Hongrie, (october 1985).

[KOR 84] KORACH E., MORAN S., ZAKS S.
Tight lower and upper bounds for some distributed algorithms for a complete network of processors.
Proc. of the 3 rd ACM conf. on principles of distributed computing (august 1984), pp.199-207.

[LEL 78] LE LANN G.
Algorithms for data sharing system which use tickets.
Proc. 3rd Berkeley Workshop on Distributed Data Base and computer Network, (august 1978), pp.259-272.

[MIS 82] MISRA J., CHANDY K.M.
Termination Detection of Diffusing Computations in CSP.
ACM Toplas, Vol. 4,1, (january 1982), pp37-43.

[MIC 82] MISRA J., CHANDY K.M
A Distributed Graph Algorithm : Knot Detection.
ACM TOPLAS, Vol. 4,4, (october 1982), pp.678-680.

[PAC 84] PACHL J.A., KORACH E., ROTEM D.
Lower Bounds for Distributed Maximum Finding Algorithms.
Journal of the ACM, Vol.31,4, (october 1984), pp.905-918.

[PET 82] PETERSON G.L.
An O(nlogn) Unidirectionnal Algorithm for the Circular Extrema Problem.
ACM TOPLAS, Vol. 4,4, (october 1982), pp.758-762.

[RAY 85] RAYNAL M.
Algorithmes Distribués et protocoles.
Eyrolles Ed., (Septembre 1985), 144p.

[SCH 84] SCHNEIDER F.D., GRIES D., SCHLICHTING R.
Fault Tolerant Broadcasts.
Science of Programming, Vol. 4,1, (1984) pp.1-15.

[SEG 83] SEGALL A.
Distributed Network Protocols.
IEEE Trans. on Inf. Theory, Vol. IT29,1, (january 1983), pp.23-35.

ANNEXE 1 : DEMONSTRATIONS

Lemme 1.

M1 :Le premier message de poids k reçu par un processus P_i ($i \neq k$) est nécessairement de type *explorer*.

Démonstration

Si P_i reçoit un message *rebrousser* de poids k, à l'étape t, ce message provient d'un de ses voisins P_j. Ce dernier processus a émis ce message à l'étape $t-1$, son contexte local vérifiant nécessairement :

(1) $pred_i = i$

(2) $pgvu_i = k$

Mais les variables $pred_i$ et $pgvu_i$ ne sont mises à jour que sur réception d'un message *explorer* de poids supérieur à $pgvu_i$, donc :

(1) $\Rightarrow$ à une étape $t' \leq t-1$, P_j a reçu un message *explorer* (sans quoi $pred_j = nil$), et le message *explorer* de plus fort poids reçu jusqu'alors provient de P_i,

(2) $\Rightarrow$ ce poids maximum, ayant atteint P_j, est égal à k.

Ainsi, à une étape $t'' < t-1$, P_i a envoyé un message *explorer* de poids k à P_j. Mais ceci implique, si $i \neq k$:

. d'une part $i < k$

. d'autre part, lors de cette étape, $pgvu_i = k$. P_i a donc nécessairement reçu, à une étape $t''' < t''$, un message *explorer* de poids k

QED.

Lemme 2

Pour tout message (*explorer* ou *rebrousser*) d'une exploration de poids k, les champs z et s verifient :

P1 : z est l'ensemble des processus ayant reçu un message de poids k et n'ayant pas éteint l'exploration.

P2 : $k = maximum(z)$.

Q1 : $s \cap z = \emptyset$.

Q2 : le destinataire du message n'appartient pas à s.

Démonstration

(i) Le premier message de l'exploration de poids k est de type *explorer*, avec $z = \{k\}$, $s = voisins_k - \{x\}$ où x est le destinataire . Comme un processus n'appartient pas à l'ensemble de ses voisins, les 4 propriétés sont vérifiées.

(ii) Soit un message *explorer(k,z,s)* émis par P_j vers P_i , et vérifiant P1, P2, Q1, Q2 (hypothèse de récurrence). Examinons les 4 situations possibles à la réception de ce message par P_i (étape courante).

(a) P_i éteint l'exploration : aucun autre message de poids k n'est émis, donc le lemme est démontré.

(b) P_i conclut l'exploration : même situation qu'en a)

(c) P_i émet *rebrousser*(k,z',s') *vers* P_j où $j = pred_i$
et (1) $z' = z \cup \{i\}$, (2) $s' = s$.
(1) ⇒ P1 reste vérifiée. D'autre part, $i \leq pgvu_i < k$
donc P2 reste vérifiée.
En outre : $s' \cap z' = (s \cap z) \cup (s \cap \{i\})$ mais $s \cap z = \emptyset$
(hypothèse de récurrence Q1) et $i \notin s$ (hypothèse Q2)
donc $s' \cap z' = \emptyset$ et Q1 reste vérifiée.
Enfin, $j \in z$ donc $j \notin s$ (hypothèse Q1) et Q2 reste vérifiée.

(d) P_i émet *explorer*(k,z',s') vers P_x, avec :
(1) $z' = z \cup \{i\}$
(2) $s' = s \cup (voisins_i - z - \{x\}.)$,
(1) ⇒ P1 reste vérifiée. D'autre part, $i \leq pgvu_i < k$ donc P2 reste vérifiée.
En outre, $s' \cap z' = (s \cap z) \cup ((voisins_i - z - \{x\}) \cap z)$
$\cup (s \cap \{i\})$
$\cup ((voisins_i - z - \{x\}) \cap \{i\})$
mais $s \cap z = \emptyset$ (hypothèse Q1)
$voisins_i - z - \{x\} \cap z = \emptyset$ *(construction)*
$i \notin s$ (hypothèse Q2)
$voisins_i - z - \{x\}) \cap \{i\} = \emptyset$ *car* $voisins_i - z - \{x\} \subset voisins_i$
et $i \notin voisins_i$
d'où $s' \cap z' = \emptyset$: ce qui montre Q1.
Enfin, par construction, $x \notin s'$ ce qui montre Q2.

(iii) Soit un message *rebrousser(k,z,s)* émis par P_j vers P_i ($i = pred_j$) et vérifiant P1,P2,Q1 et Q2. La démonstration est similaire à celle du (ii) ci-dessus : on examine les situations possibles lorsque P_i reçoit ce message (étape courante) :

(a) $pgvu_i > k$: exploration éteinte

(b) P_i émet *rebrousser*(k,z',s') vers P_l où $l = pred_i$, avec $z' = z$ $s' = s$. D'après le lemme 1, $i \in z$ donc P1 est vérifié et, comme $i \leq pgvu_i = k$, P2 reste vérifiée.
Enfin, $s' \cap z' = s \cap z = \emptyset$ (Q1) et d'après le lemme 1 $l \in z$ d'où $l \notin s = s'$ (Q2).

(c) P_i émet *explorer*(k,z',s') vers P_x, avec $z' = z$, $s' = s - \{x\}$. D'après le lemme 1, $i \in z$ donc P1 est vérifiée, et comme $i \leq pgvu_i = k$ il en est de même de P2. Enfin, $s' \cap z' \subset s \cap z = \emptyset$ d'où Q1. Par construction, $x \notin s'$ d'où Q2. QED.

Lemme 3

(M2) P_k ne peut recevoir aucun message *explorer* de poids k.

(M3) $\forall\ i \neq k$, P_i reçoit au plus un message *explorer* de poids k.

Démonstration

Lorsque P_k lance l'exploration de poids k, il est mémorisé dans l'attribut z, transporté par les messages de l'exploration.

Si $i \neq k$, lorsque P_i reçoit un message *explorer* de poids k, ou bien il éteint ou conclut l'exploration, ou bien il est mémorisé dans l'attribut z.

D'autre part, un message *explorer* n'est envoyé vers un processus que si celui-ci n'est pas déjà visité (ligne 1) ou appartient à s (ligne 2) c'est à dire, d'après le lemme 2, Q1, n'appartient pas à z. QED.

Lemme 4

(i) A1 : Pour une exploration de poids k la propriété suivante est vérifiée à chaque étape : le sous-graphe partiel A_k est une arborescence de racine P_k, dans laquelle, pour tout P_i vérifiant $pgvu_i = k$:

.$pred_i$ est le père de P_i

. $succ_i$ est l'ensemble des fils de P_i.

(ii) Pour tout message (*explorer* ou *rebrousser*) émis par P_j vers P_i, les champs s et z vérifient :

P3 : s est égal à l'ensemble des voisins non atteints des processus situés sur γ_{kj}.

P4 : *Si* $l \in z$ et si P_l possède des voisins non atteints alors P_l est situé γ_{kj}.

Démonstration

On procède par récurrence, comme pour 2.

. Lorsque P_k lance l'exploration de poids k, on a initialement :

$X_k = \{P_k\}$, $\Gamma_k = \emptyset$ et comme $état_i = initial$, $pred_k = nil$, $succ_k = \emptyset$

Le premier message est de type *explorer*, à destination de P_x, avec $z = \{k\}$, $s = voisins_k - \{x\}$ P3 et P4 sont vérifiées.

. Supposons le lemme démontré jusqu'à l'étape t, et, si celle-ci n'est pas une étape d'extinction ou de conclusion, pour le message μ émis à cette étape.

Soient z, k, s les champs de ce message, destiné à P_i. Nous montrons que :

(i) à l'étape courante (réception de μ par P_i), la propriété A1 reste vérifiée.

(ii) Si cette étape provoque l'émission d'un nouveau message , P3 et P4 restent vérifiées. Pour cela, nous examinons les différentes situations de l'étape courante :

* étape d'extinction :

A_k n'est pas modifié lors de cette étape, et l'exploration est stoppée. Donc le lemme est démontré.

* étape de conclusion :

μ est un message de type *explorer* et le sommet P_i et l'arc (P_j, P_i) sont ajoutés à A_k. D'après M3, μ est le premier message de poids k reçu par P_i donc le chemin de P_k à P_i est unique ($\gamma_{ki} = \gamma_{kj}.(j,i)$).
En outre, $pred_i = j$, et i a été rangé dans $succ_j$ lorsque P_j a émis μ.
Enfin, aucun nouveau message n'est émis.

* étape de génération:

Si μ est de type *explorer*, alors A1 se montre comme dans le cas précédent (étape de conclusion). D'autre part, P_i emet *explorer*(k,z',s') vers P_x, avec $z' = z \cup \{i\}$, $s' = s \cup (voisins_i - z - \{x\})$.

D'après l'hypothèse de récurrence P3, et la définition de s', ce dernier ensemble est bien égal à l'ensemble des voisins non atteints de P_i et de ses ascendants puisque

$$\gamma_{ki} = \gamma_{kj} \cdot (j,i)$$

Enfin si $P_l \in z'$ et possède des voisins non atteints, alors

- ou bien $P_l \in z \Rightarrow P_l$ est sur γ_{kj} (hypothèse de récurrence P4)
- ou bien $P_l = P_i$

donc P_l est sur γ_{ki}

Si μ est de type *rebrousser*, A_k n'est pas modifié à l'étape courante, ni la variable $pred_i$ à la réception de μ, ni $succ_j$ lorsque P_j a envoyé μ (P_i étant déjà dans A_k); de plus $\gamma_{ki} = \gamma_{kj} - (i,j)$. D'autre part, P_i émet *explorer*(k,z',s') *vers* P_x, avec $z' = z$ et $s' = s - \{x\}$.

D'après l'hypothèse P3, s est l'ensemble des voisins non atteints de P_i et de ses ascendants dans A_k. s' vérifie donc bien P3 (P_i est l'emetteur du message, et P_x devient un voisin atteint de P_i).

Enfin si $P_l \in z'$ et possède des voisins non atteints, alors P_l est sur γ_{kj} ($z' = z$ et hypothèse de récurrence P4).

Mais $voisins_j \subset z$ (μ, émis par P_j, est de type *rebrousser*, donc P_l est sur $\gamma_{kj} - (i,j) = \gamma_{ki}$

* étape de renvoi :

Si μ est de type *explorer*, alors on montre A1 comme pour les deux étapes précédentes (étape de conclusion ou de génération). D'autre part, $voisins_i \subset z$ et P_i envoie *rebrousser*(k,z',s') à P_j, où $j = pred_i$, avec $z' = z \cup \{i\}$, $s' = s$.

D'après l'hypothèse de récurrence P3, s est l'ensemble des voisins non atteints des processus situés sur γ_{kj}.

s' est donc l'ensemble des voisins non atteints des processus situés sur γ_{ki}, puisque $voisins_i \subset z$ et $\gamma_{ki} = \gamma_{kj} \cdot (j,i)$.

Enfin, si $P_l \in z'$ et possède des voisins non atteints, alors $P_l \in z$ (puisque $voisins_i \subset z$) d'où P_l est sur γ_{kj} (hypothèse de récurrence P4)

et comme $\gamma_{kj} \subset \gamma_{ki}$, P4 rest vérifiée.

Si μ est de type *rebrousser* alors A1 se montre comme pour l'étape de génération. D'autre part, $voisins_i \cap s = \emptyset$ donc $voisins_i \subset z$. P_i envoie à P_l, où $l = pred_i$, un message $rebrousser(k,z',s')$, avec

$$z' = z, \qquad s' = s .$$

D'après l'hypothèse P3, s est l'ensemble des voisins non atteints des processus situés γ_{kj}; mais comme $voisins_i \subset z$ c'est aussi l'ensemble des voisins non atteints des processus situés sur γ_{ki}. Ce qui, avec $s' = s$, montre P3.

Enfin, P4 se démontre comme dans le cas de l'étape de génération, sur réception d'un message de type *rebrousser*.

ANNEXE 2 : TOPOLOGIES PARTICULIERES.

On étudie ici le nombre de messages nécessaires à l'algorithme en profondeur d'abord dans deux cas particuliers : le maillage complet et l'anneau.

1) réseau complet

Il va de soi qu'aucun des processus ne sait que le maillage est complet, l'algorithme étant indépendant de toute topologie; (sinon aucun message n'est nécessaire pour le calcul , chaque processus ayant le nombre et l'identite de tous les autres peut en calculer le maximum; les seuls messages nécessaires étant alors un signal indiquant à chacun de considérer le maximum).

Dans ce cas seul P_{max} est maximum local. Donc, si tous les processus démarrent en même temps (cas où l'on obtient le plus de messages), on aura :

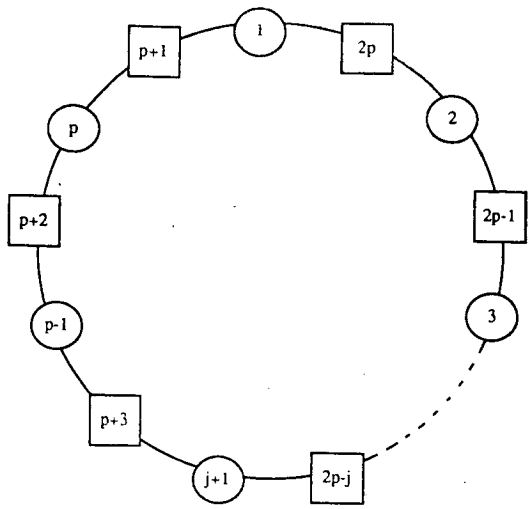
- $n - 1$ messages *explorer* (exploration de poids k, $k < max$)
- $n - 1$ messages *explorer* (exploration de poids *max*; l'arborescence est alors réduite à un chemin sur lequel les sommets sont ordonnés par ordre de poids décroissant; c'est le plus petit sommet qui conclut).
- $n - 1$ messages *conclure*

soit 3(n-1) messages; la complexité est alors O(n).

2) anneau

Nous donnons un exemple pour lequel une complexité en n^2 est effectivement atteinte :

Il y a $n = 2p$ processus rangés en anneau avec l'ordre suivant sur leur poids respectif



Les deux sous-suites, de longueur p chacune :

$\alpha = 2p,\ 2p-1,\ldots,2p-j,\ldots,p+2,\ 2p-(p-1) = p+1$ (décroissante)

$\beta = 1,\ 2,\ 3,\ \ldots,\ j+1,\ \ldots,\ p$ (croissante)

sont donc entrelacées.

Supposons que tous les processus démarrent simultanément.

Ceux de la sous-suite β ne sont pas des maxima locaux, donc chacun n'engendre qu'un message *explorer* soit au total p messages *explorer*.

Examinons le nombre de messages *explorer* engendré par chacun des processus de la sous-suite α (en supposant les vitesses de communication égales). Rappelons qu'un processus émettant pour la première fois un message relatif à une exploration donnée l'envoie à celui de ses voisins doté de la plus grande identité.

Un message *explorer* de poids $p+1 = 2p-(p-1)$ part vers le processus P_p et s'éteint sur le processus P_{p+2} ce qui génère 2 messages.

De la même manière un message *explorer* de poids $p+2 = 2p-(p-2)$ part de P_{p+2} vers P_p et s'éteint sur P_{2p} ce qui génère 4 messages.

De manière générale sur cet anneau, un message *explorer* de poids $p+(p-j) = 2p-j$ part de P_{2p-j} vers P_{j+2} et s'éteint sur P_{2p} ce qui génère $2(p-j)$ messages

On a finalement $2+4+6+,\ldots,+2(p-1)+(2p-1)$ messages générés au pire sur cet anneau.

Soit : $2(1+2+\ldots+p)-1 = p(p+1)-1$

comme $p = n/2$, on a donc :

$$n/2\ (n/2+1)-1 = O(n^2).$$

Aucun message de type *rebrousser* n'est nécessaire dans une structure en anneau (cf. paragraphe 4.3); par contre n-1 messages de type *conclure* sont nécessaires ce qui donne finalement une complexité $O(n^2)$.

PI 280 **Commande de systèmes redondants et évitement d'obstacles**
Bernard Espiau – 52 pages ; Janvier 86.

PI 281 **A distributed algorithm for mutual exclusion in an arbitrary network**
Jean-Michel Hélary, Noël Plouzeau, Michel Raynal – 16 pages ; Janvier 86.

PI 282 **Stabilité robuste dans la commande adaptative indirecte passive**
Philippe de Larminat – 70 pages ; Janvier 86.

PI 283 **Data synchronized pipeline architecture pipelining in multiprocessor environments**
Yvon Jégou, André Seznec – 36 pages ; Janvier 86.

PI 284 **An overview of the GOTHIC Distributed Operating System**
Jean-Pierre Banatre, Michel Banatre, Florimond Ployette – 24 pages ; Janvier 86.

PI 285 **Optimal sensor location for detecting changes in dynamical behavior**
Michèle Basseville, Albert Benveniste, Georges Moustakides, Anne Rougée – 32 pages ; Février 86.

PI 286 **La tolérance aux fautes dans un système temps-réel à contraintes strictes**
Maryline Silly – 32 pages ; Février 86.

PI 287 **A new statistical approach for the automatic segmentation of continuous speech signals**
Régine André-Obrecht – 38 pages ; Mars 86.

PI 288 **Synthèse sur les réseaux locaux temps-réel**
Philippe Belmans – 40 pages ; Mars 86.

PI 289 **Calcul distribué d'un extrémum et du routage associé dans un réseau quelconque**
Jean-Michel Hélary, Aomar Maddi, Michel Raynal – 36 pages ; Mars 86.

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique